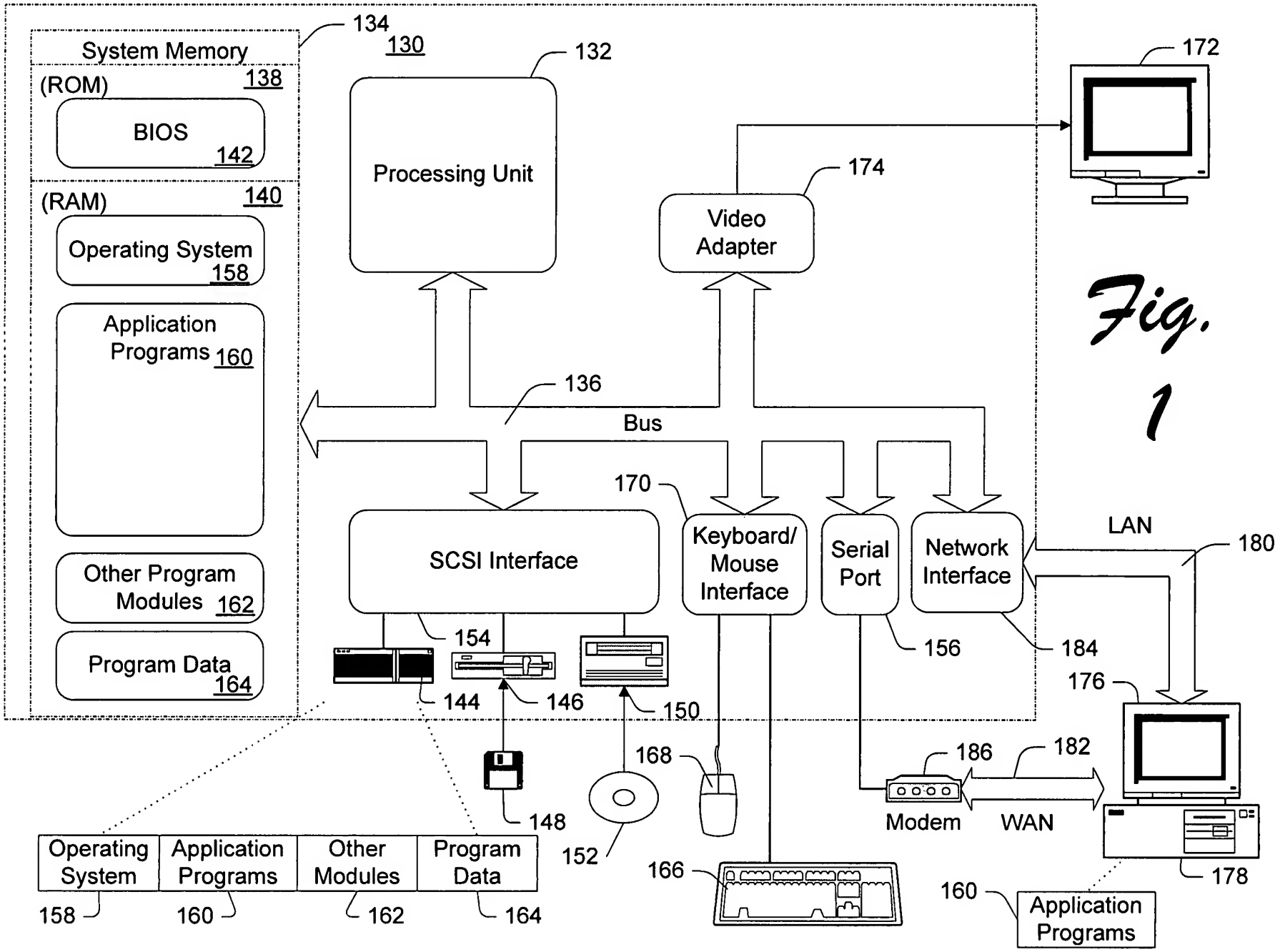
System Memory
134
130
(ROM)
138
BIOS
142
(RAM)
140
Operating System
158
Application Programs
160
Other Program Modules
162
Program Data
164
Processing Unit
132
Video Adapter
174
172
136
Bus
170
SCSI Interface
Keyboard/ Mouse Interface
Serial Port
Network Interface
LAN
180
154
144
146
150
156
184
176
168
186
182
148
152
Modem
WAN
166
178
Operating System
Application Programs
Other Modules
Program Data
158
160
162
164
160
Application Programs

Fig. 1

200

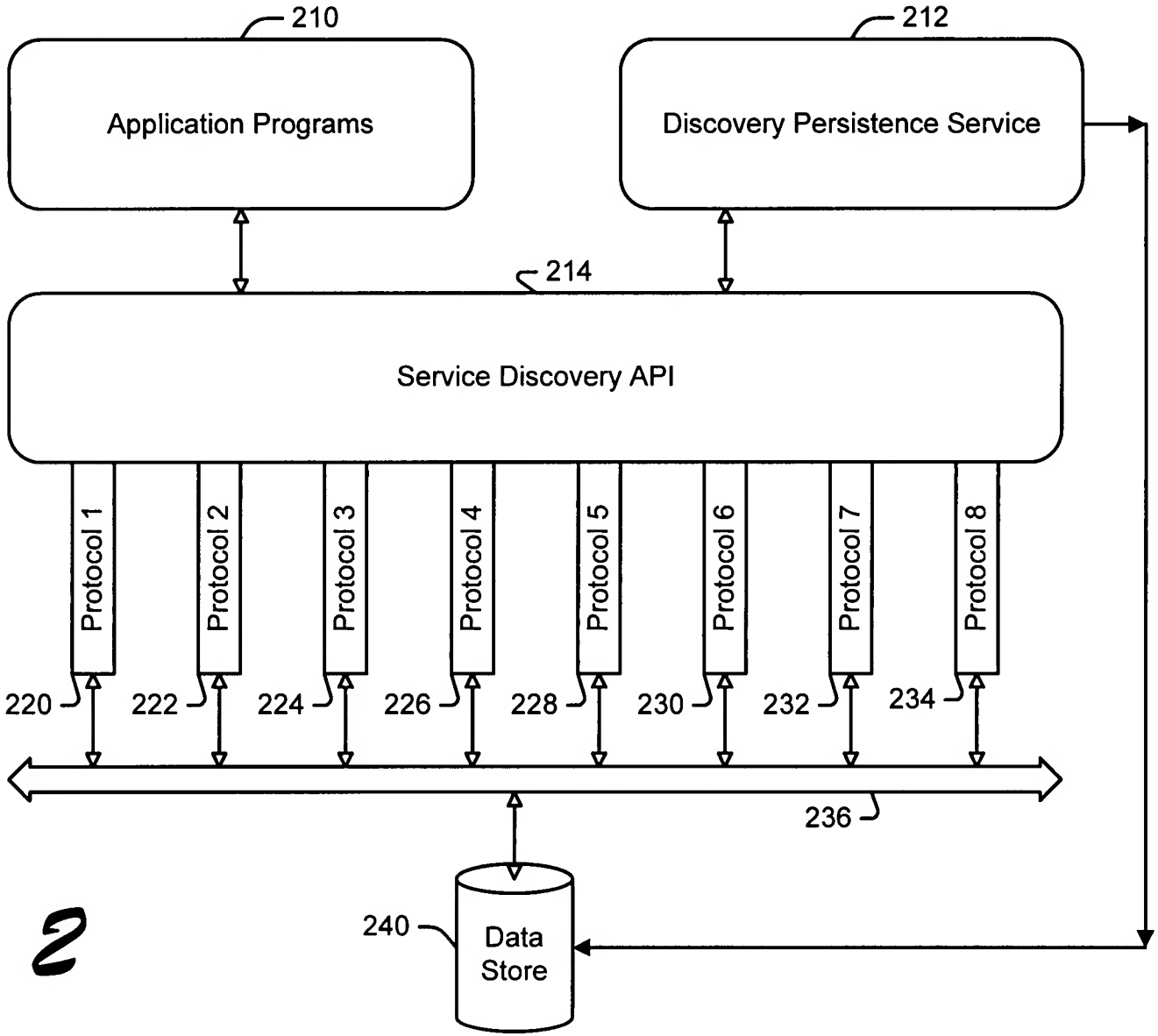
210
Application Programs
212
Discovery Persistence Service
214
Service Discovery API
Protocol 1
Protocol 2
Protocol 3
Protocol 4
Protocol 5
Protocol 6
Protocol 7
Protocol 8
220
222
224
226
228
230
232
234
236
240
Data Store

Fig. 2

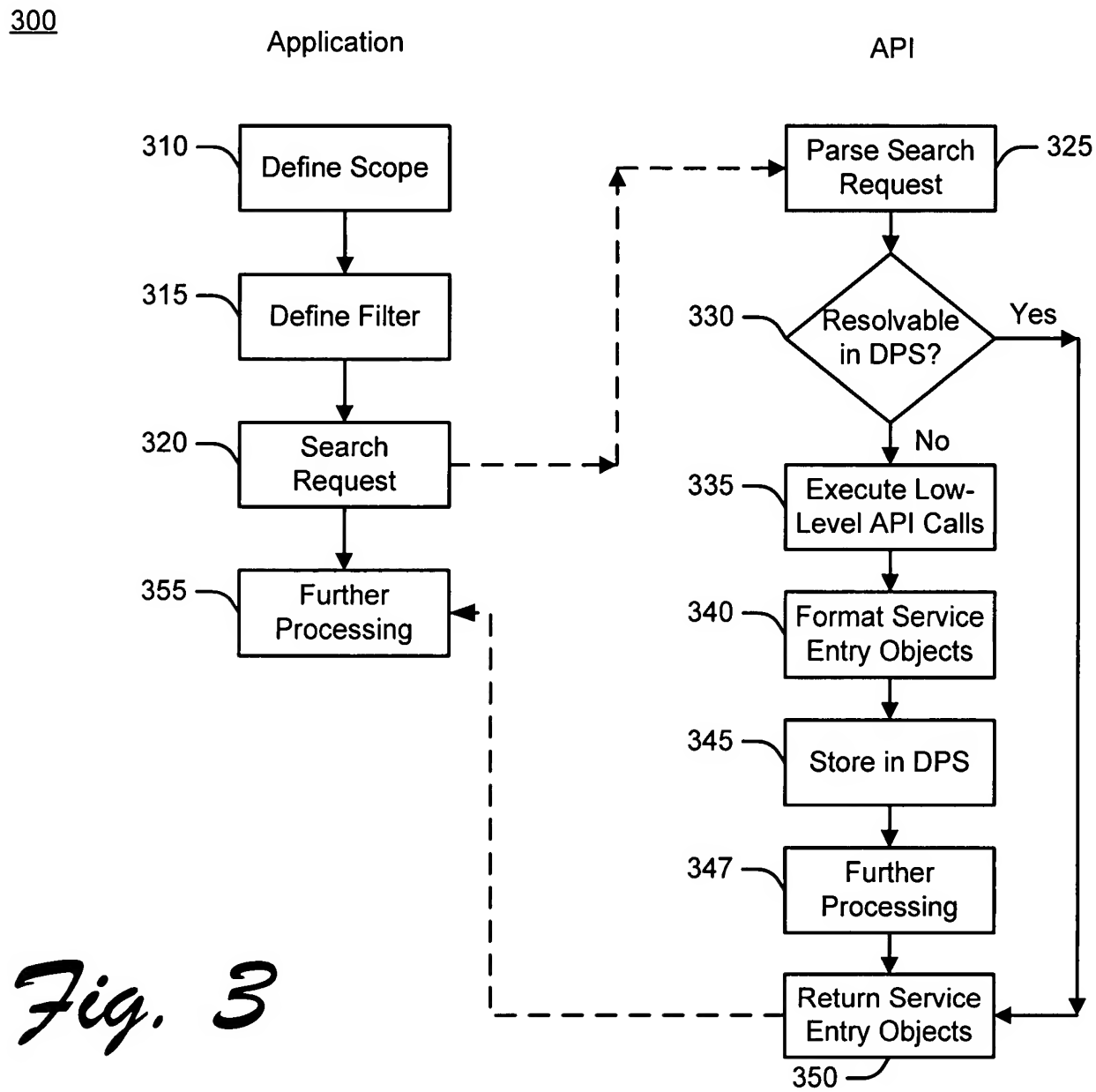
300
Application
API
310
Define Scope
315
Define Filter
320
Search Request
355
Further Processing
325
Parse Search Request
330
Resolvable in DPS?
Yes
No
335
Execute Low-Level API Calls
340
Format Service Entry Objects
345
Store in DPS
347
Further Processing
Return Service Entry Objects
350

Fig. 3

400

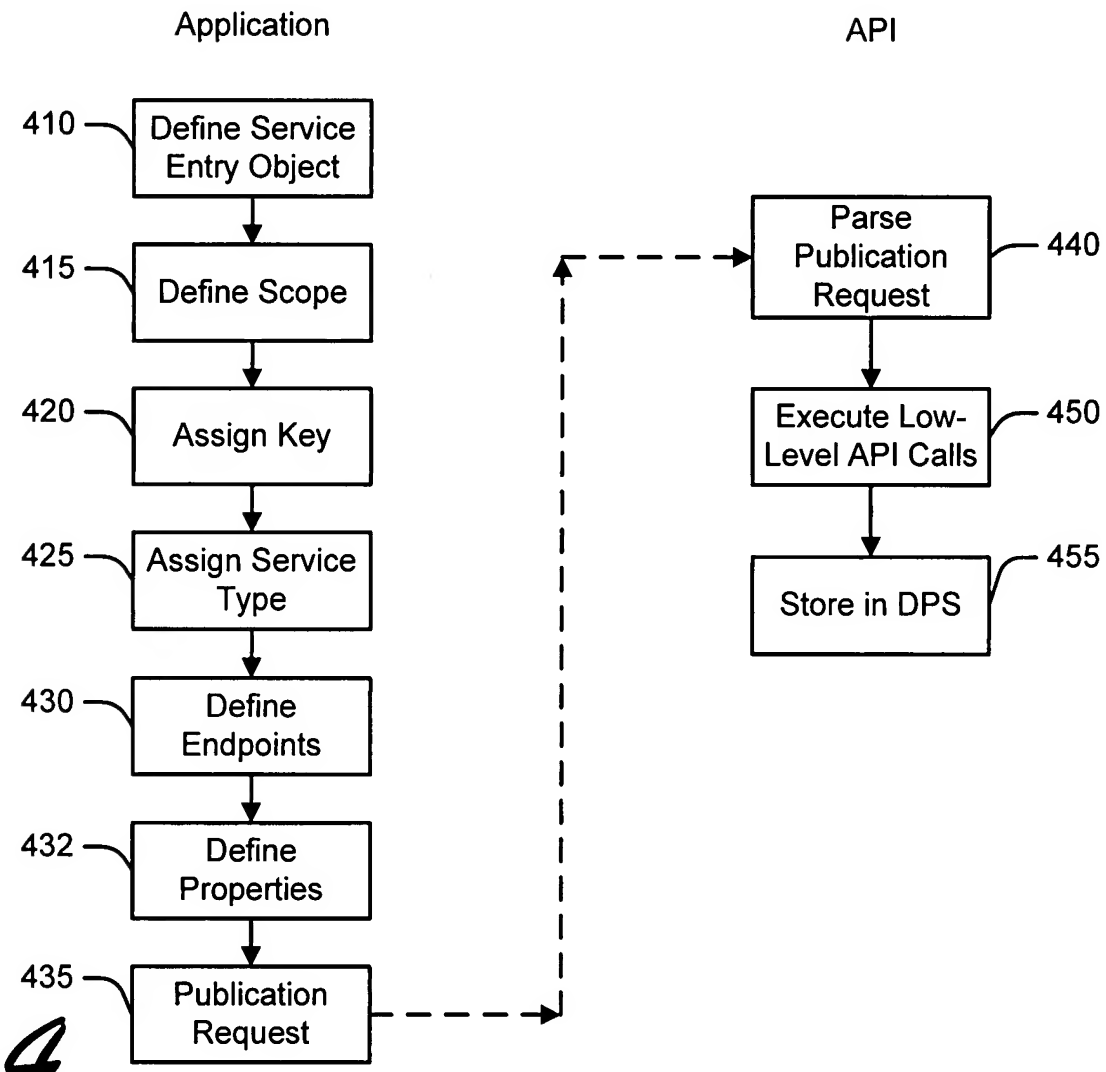
Application
API
410
Define Service Entry Object
415
Define Scope
420
Assign Key
425
Assign Service Type
430
Define Endpoints
432
Define Properties
435
Publication Request
Parse Publication Request
440
Execute Low-Level API Calls
450
Store in DPS
455

Fig. 4

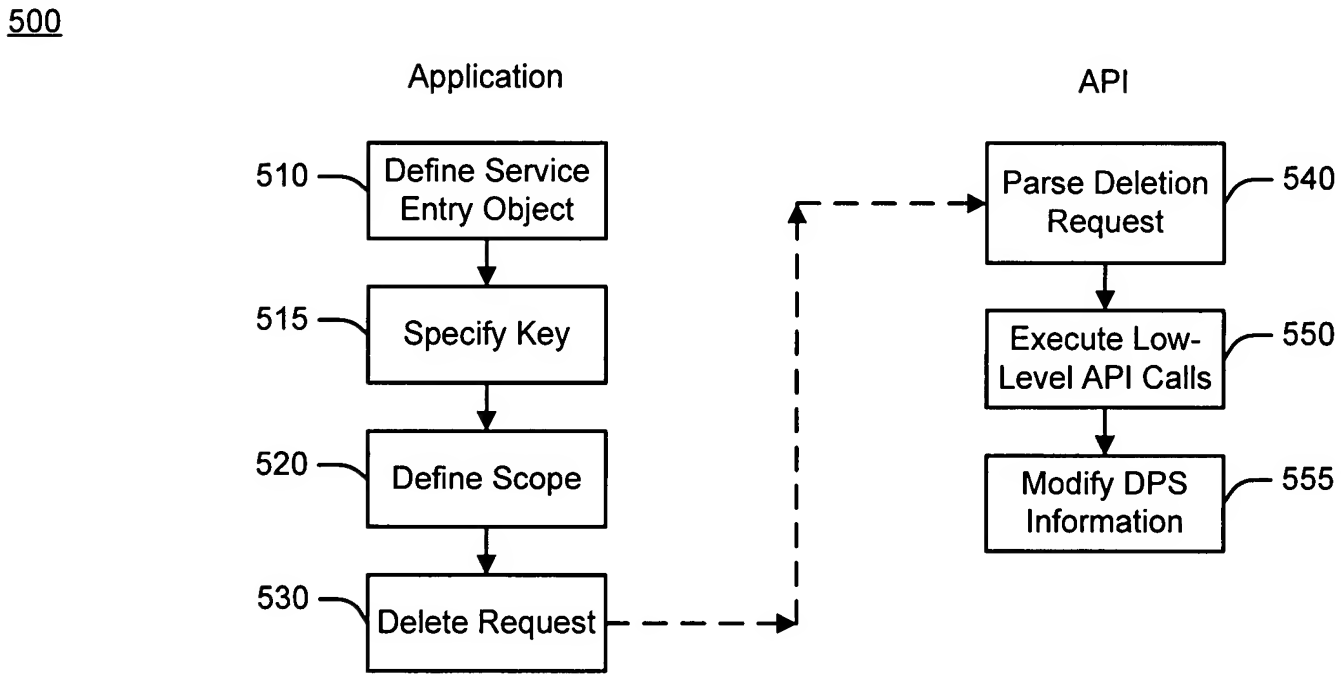
500
Application
API
510
Define Service Entry Object
515
Specify Key
520
Define Scope
530
Delete Request
Parse Deletion Request
540
Execute Low-Level API Calls
550
Modify DPS Information
555

Fig. 5

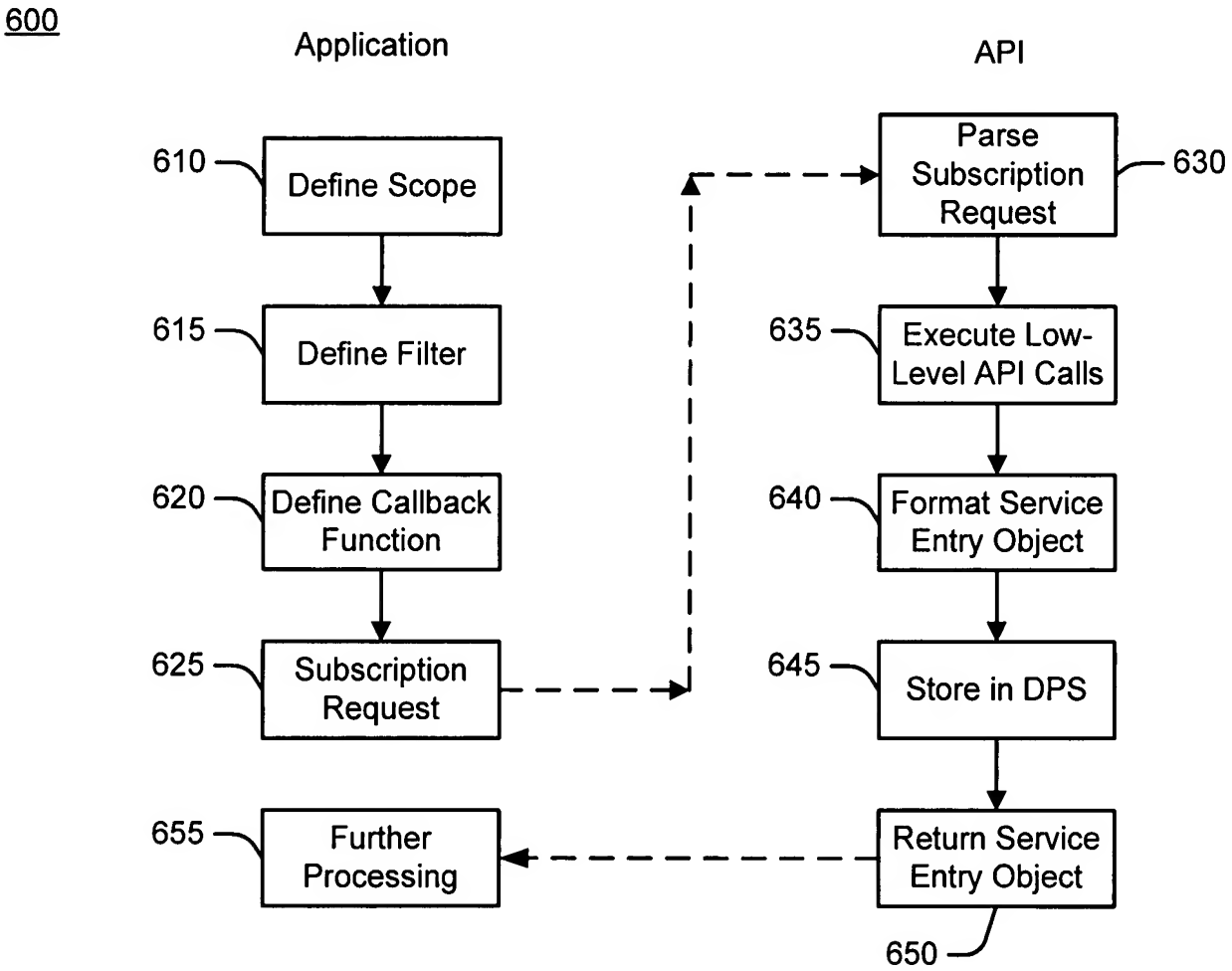
600
Application
API
610
Define Scope
615
Define Filter
620
Define Callback Function
625
Subscription Request
655
Further Processing
Parse Subscription Request
630
635
Execute Low-Level API Calls
640
Format Service Entry Object
645
Store in DPS
Return Service Entry Object
650

Fig. 6

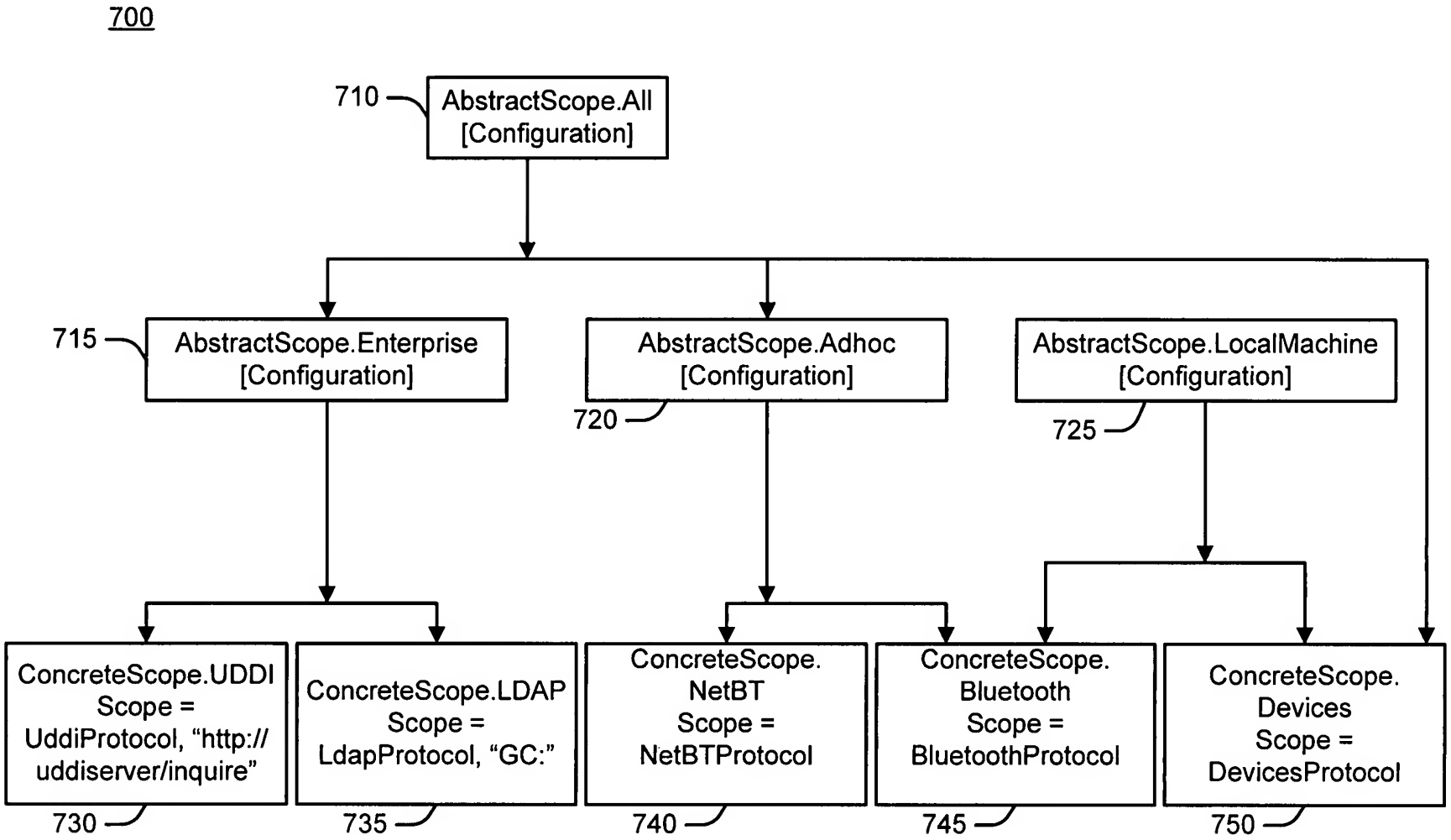
700
710
AbstractScope.All
[Configuration]
715
AbstractScope.Enterprise
[Configuration]
720
AbstractScope.Adhoc
[Configuration]
725
AbstractScope.LocalMachine
[Configuration]
ConcreteScope.UDDI
Scope =
UddiProtocol, "http://
uddiserver/inquire"
730
ConcreteScope.LDAP
Scope =
LdapProtocol, "GC:"
735
ConcreteScope.
NetBT
Scope =
NetBTProtocol
740
ConcreteScope.
Bluetooth
Scope =
BluetoothProtocol
745
ConcreteScope.
Devices
Scope =
DevicesProtocol
750

Fig. 7

```
static void SimpleFilterAD()
{
   // Declare a ServiceFinder
   System.Discovery.ServiceFinder servFind = new ServiceFinder();

   // Specify to search using Active Directory in the global catalog
   servFind.Scopes.Add (new ConcreteScope (ConcreteScope.ADProtocol, "GC:", null));

   // Filter for only printers
   SimpleFilter filter = new SimpleFilter ();
   filter.ServiceType = CommonServiceTypes.Printer;
   servFind.Filter = filter;

   // Declare the properties to match on
   filter.Properties.Add ("printcolor", "TRUE");
   filter.Properties.Add ("pagesperminute", "50");

   // Find all entities matching the filter within the scope
   ServiceEntryCollection seColl = servFind.FindAll ();

   // Report results
   Console.WriteLine ("Results Found: " + seColl.Count);

   foreach (ServiceEntry se in seColl)
   {
      Console.WriteLine ("Name: " + se.Name);
      Console.WriteLine ("Description: " + se.Description);
      If ((se.Endpoints!=null) && (se.Endpoints.Count>0))
      {
         Console.WriteLine ("Address: " + se.Endpoints[0].Address);
      }
      Console.WriteLine ("Key: " + se.Key);
      Console.WriteLine ();
   }
}
```

Fig. 8

```
static void RichFilterUDDI ()
{
   // Declare a ServiceFinder
   ServiceFinder servFind = new ServiceFinder ();

   // Specify to search using the UDDI protocol in the
   // test version of the Microsoft UDDI Business Registry
   servFind.Scopes.Add (new ConcreteScope (ConcreteScope.UddiProtocol,
                  "http://test.uddi.microsoft.com/inquire", null));

   // Filter for only Web services named Fabrikam
   // that implement a specific interface i.e. the
   // tModel "uddi-org:inquiry_v2"
   RichFilter filter = new RichFilter
      ("WebService[ name = 'Fabrikam' and ServiceInterface = 'uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b' ]");

   servFind.Filter = filter;

   // Find all entities matching the filter within the scope
   ServiceEntryCollection seColl = servFind.FindAll ();

   // Report results
   Console.WriteLine ("Results Found: " + seColl.Count);
   foreach (ServiceEntry se in seColl)
   {
      Console.WriteLine ("Name: " + se.Name);
      if((se.Endpoints!=null) && (se.Endpoints.Count>0))
      {
         Console.WriteLine ("Address: " + se.Endpoints[0].Address);
      }
      Console.WriteLine ("Key: " + se.Key);
      Console.WriteLine ();
   }
}
```

Fig. 9

```
static void SimpleFilterUDDI ()
{
        // Declare a ServiceFinder.
        System.Discovery.ServiceFinder servFind = new ServiceFinder ();

        // Specify to search using the UDDI protocol in the
// Microsoft UDDI Business Registry node.
        servFind.Scopes.Add (new ConcreteScope (ConcreteScope.UddiProtocol,
"http://uddi.microsoft.com/inquire", null));

        // Filter for only services that implement a specific interface,
// i.e. the tModel "uddi-org:inquiry_v2".
        SimpleFilter filter = new SimpleFilter ();
        filter.ServiceInterfaces.Add(
"uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b");

        servFind.Filter = filter;

        // Find all entities matching the filter within the scope.
        ServiceEntryCollection seColl = servFind.FindAll ();

        // Report results.
        Console.WriteLine ("Results Found: " + seColl.Count);

        foreach (ServiceEntry se in seColl)
        {
                Console.WriteLine ("Name: " + se.Name);
                Console.WriteLine ("Description: " + se.Description);
                if((se.Endpoints!=null) && (se.Endpoints.Count>0))
                {
                        Console.WriteLine ("Address: " + se.Endpoints[0].Address);
                }
                Console.WriteLine ("Key: " + se.Key);
        }
}
```

Fig. 10

```
Sub SimpleFilterUDDI()
    ' Declare a ServiceFinder.
    Dim servFind = New ServiceFinder

    ' Specify to search using the UDDI protocol in the
    ' Microsoft UDDI Business Registry node.
    servFind.Scopes.Add(New ConcreteScope(ConcreteScope.UddiProtocol, _
"http://uddi.microsoft.com/inquire", Nothing))

    ' Filter for only services that implement a specific interface,
    ' i.e. the tModel "uddi-org:inquiry_v2".
    Dim filter As New SimpleFilter
    filter.ServiceInterfaces.Add( _
"uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b")

    servFind.Filter = filter

    ' Find all entities matching the filter within the scope.
    Dim seColl As ServiceEntryCollection = servFind.FindAll()

    ' Report results.
    Console.WriteLine("Results Found: " & seColl.Count)

    Dim se As ServiceEntry
    For Each se In seColl
        Console.WriteLine("Name: " & se.Name)
        Console.WriteLine("Description: " & se.Description)
        If Not (se.Endpoints Is Nothing) AndAlso se.Endpoints.Count > 0 Then
            Console.WriteLine("Address: " & se.Endpoints(0).Address)
        End If
        Console.WriteLine("Key: " & se.Key)
    Next se

End Sub 'SimpleFilterUDDI
```

Fig. 11

```
static void RichFilterAD ()
{
        // Declare a ServiceFinder.
        ServiceFinder servFind = new ServiceFinder ();

        // Specify to search using Active Directory in the global catalog.
        servFind.Scopes.Add (new ConcreteScope (
ConcreteScope.ADProtocol, "GC:", null));

        // Filter for printers where the name begins with 'Office Printer'.
        RichFilter filter = new RichFilter (
"Printer[ name like 'Office Printer' ]");

        servFind.Filter = filter;

        // Find all entities matching the filter within the scope.
        ServiceEntryCollection seColl = servFind.FindAll ();

        // Report results
        Console.WriteLine ("Results Found: " + seColl.Count);
        foreach (ServiceEntry se in seColl)
        {
                Console.WriteLine ("Name: " + se.Name);
                if((se.Endpoints!=null) && (se.Endpoints.Count>0))
                {
                        Console.WriteLine ("Address: " + se.Endpoints[0].Address);
                }
                Console.WriteLine ("Key: " + se.Key);
        }
}
```

Fig. 12

```
Sub RichFilterAD()
    ' Declare a ServiceFinder.
    Dim servFind As New ServiceFinder

    ' Specify to search using Active Directory in the global catalog.
    servFind.Scopes.Add(New ConcreteScope( _
            ConcreteScope.ADProtocol, "GC:", Nothing))

    ' Filter for printers where the name begins with 'Office Printer'.
    Dim filter As New RichFilter("Printer[ name like 'Office Printer' ]")

    servFind.Filter = filter

    ' Find all entities matching the filter within the scope.
    Dim seColl As ServiceEntryCollection = servFind.FindAll()

    ' Report results
    Console.WriteLine("Results Found: " & seColl.Count)
    Dim se As ServiceEntry
    For Each se In seColl
        Console.WriteLine("Name: " & se.Name)
        If Not (se.Endpoints Is Nothing) AndAlso se.Endpoints.Count > 0 Then
            Console.WriteLine("Address: " & se.Endpoints(0).Address)
        End If
        Console.WriteLine("Key: " & se.Key)
    Next se

End Sub 'RichFilterAD
```

Fig. 13

```
const string sampleServiceType = "75FF48AB-E771-4d44-9E8C-E09141112417";
static string sampleServiceKey = Guid.NewGuid().ToString();

static void Save()
{
        // Instantiate a new ServiceEntry object.
        ServiceEntry service = new ServiceEntry();

        // Add the scope to publish this service entry into.
        service.Scopes.Add( new ConcreteScope( ConcreteScope.SsdpProtocol ) );

        // Assign a unique key, some protocols may generate one.
        service.Key = sampleServiceKey;

        // The type of the service is specified by setting
        // the ServiceType property.
        service.ServiceType = new ServiceInterface( sampleServiceType );

        // Add an endpoint reference to the address for accessing the service.
        Endpoint ep = new Endpoint( Environment.MachineName, null );
        service.Endpoints.Add( ep );

        // Publish the service information synchronously
        // into the specified scope.
        service.Save();
}
```

Fig. 14

```
Const sampleServiceType As String = "75FF48AB-E771-4d44-9E8C-E09141112417"
Shared sampleServiceKey As String = Guid.NewGuid().ToString()

Shared Sub Save()
   ' Instantiate a new ServiceEntry object.
   Dim service As New ServiceEntry

   ' Add the scope to publish this service entry into.
   service.Scopes.Add(New ConcreteScope(ConcreteScope.SsdpProtocol))

   ' Assign a unique key, some protocols may generate one.
   service.Key = sampleServiceKey

   ' The type of the service is specified by setting
   ' the ServiceType property.
   service.ServiceType = New ServiceInterface(sampleServiceType)

   ' Add an endpoint reference to the address for accessing the service.
   Dim ep As New Endpoint(Environment.MachineName, Nothing)
   service.Endpoints.Add(ep)

   ' Publish the service information synchronously
   ' into the specified scope.
   service.Save()

End Sub 'Save
```

Fig. 15

```
static void Delete()
{
        // Instantiate a new ServiceEntry object.
        ServiceEntry service = new ServiceEntry();

        // Specify the unique key of the service to delete.
        service.Key = sampleServiceKey;

        // Add the scope to delete this service from.
        service.Scopes.Add( new ConcreteScope( ConcreteScope.SsdpProtocol ) );

        // Delete the service information synchronously
        // from the specified scope.
        service.Delete();
}
```

Fig. 16

```
Sub Delete()
    ' Instantiate a new ServiceEntry object.
    Dim service As New ServiceEntry

    ' Specify the unique key of the service to delete.
    service.Key = sampleServiceKey

    ' Add the scope to delete this service from
    service.Scopes.Add(New ConcreteScope(ConcreteScope.SsdpProtocol))

    ' Delete the service information synchronously
    ' from the specified scope.
    service.Delete()

End Sub 'Delete
```

Fig. 17

```
public sealed class SubscriptionSample
{
        // This delegate will be called each time a notification of an
// update on the SampleService occurs.
        private static ServiceFinder.ServiceUpdateEventHandler handler =
                new ServiceFinder.ServiceUpdateEventHandler( Handler );

        private static ServiceFinder finder = new ServiceFinder();

        public static void Subscribe()
        {
                // Initialize the finder to look for 'castle' home networks.
                finder.Scopes.Add( new ConcreteScope(
ConcreteScope.SsdpProtocol ) );
                finder.Filter = new SimpleFilter(
CommonServiceTypes.HomeNetwork );

                // Add our handler to the delegate for service updates.
                finder.ServiceUpdate += handler;
        }

        public static void Unsubscribe()
        {
                // Remove our handler from the delegate for service updates.
                finder.ServiceUpdate -= handler;
        }

        // This is the callback function that was registered with the delegate.
        private static void Handler( object sender, ServiceUpdateEventArgs e )
        {
                // Handle the service update here.
                Console.WriteLine( "UpdateType{0}\t(key={1})",
e.UpdateType, e.Services[ 0 ].Key );
        }
}
```

Fig. 18

```
Public NotInheritable Class SubscriptionSample

    ' This delegate will be called each time a notification of an
    ' update on the SampleService occurs.

    Private Shared finder As New ServiceFinder

    Public Shared Sub Subscribe()
        ' Initialize the finder to look for castle home networks.
        finder.Scopes.Add(New ConcreteScope(ConcreteScope.SsdpProtocol))
        finder.Filter = New SimpleFilter(CommonServiceTypes.HomeNetwork)

        ' Add our handler to the delegate for service updates.
        AddHandler finder.ServiceUpdate, AddressOf Handler
    End Sub 'Subscribe

    Public Shared Sub Unsubscribe()
        ' Remove our handler from the delegate for service updates.
        RemoveHandler finder.ServiceUpdate, AddressOf Handler
    End Sub 'Unsubscribe

    ' This is the callback function that was registered with the delegate.
    Private Shared Sub Handler(ByVal sender As Object, _
        ByVal e As ServiceUpdateEventArgs)

        ' Handle the service update here.
        Console.WriteLine("UpdateType{0}" & vbTab & "(key={1})", _
            e.UpdateType, e.Services(0).Key)
    End Sub 'Handler

End Class 'SubscriptionSample
```

Fig. 19